Distributed Authorization with CAProck

INTERPEER

@CLT2023 // Chemnitzer Linux-Tage'23

Introduction

Interpeer Project does R&D in “internet technology”;

This talk is authorization in distributed systems.

Work done under a grant from ISOC foundation.

https://www.isocfoundation.org/

What is “distributed”?

Baran, Paul (1964)
Memorandum
RM-3420-PR

Resiliency in
communications
networks

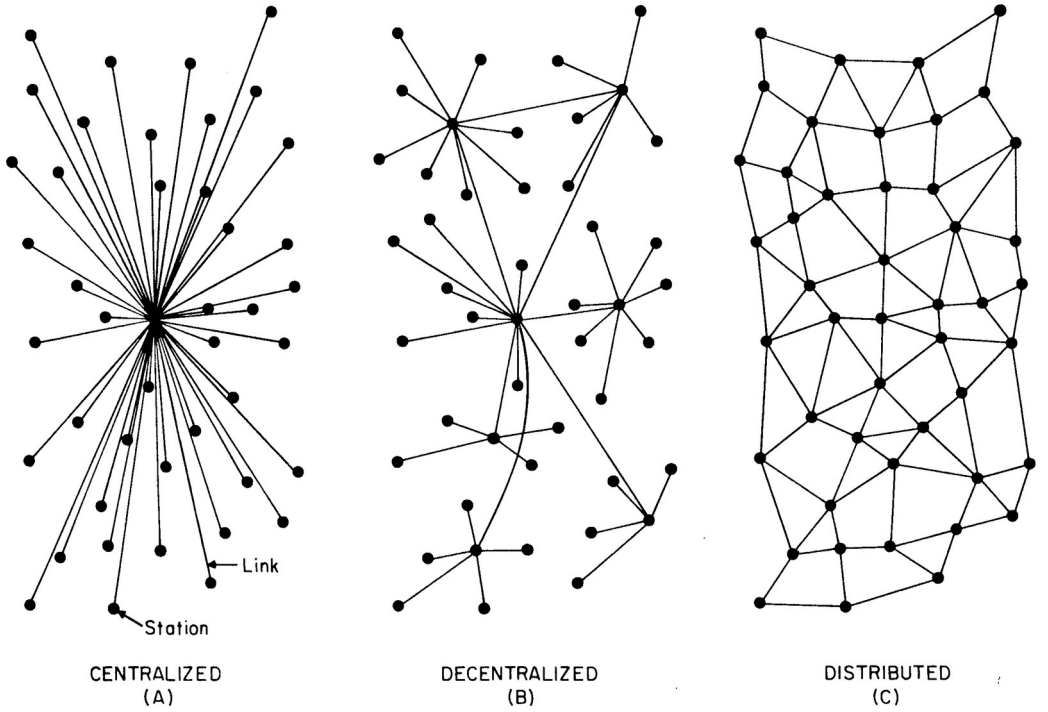


FIG. 1 — Centralized, Decentralized and Distributed Networks

Baran, P. (1964). On Distributed Communications, Memorandum RM-3420-PR.
https://www.rand.org/content/dam/rand/pubs/research_memoranda/2006/RM3420.pdf

What is “distributed”?

Distributed networks increase resiliency because every node is connected to multiple other nodes. This means individual connections can often fail without disconnecting nodes – the path may change.

This is particularly useful for mobile devices (e.g. drones, “flying cell phones”), where connectivity may fail.

Connectivity **will** fail.

Goals

How to authorize when the authority is not reachable?

Distribution is about:

1) Not relying on any single point of failure.
2) (Temporary) absence of a function as a single point of failure.

Useful for e.g.:

3) IoT provisioning
4) Offline-first use cases
5) Asynchronous use cases

Recap: Authentication vs. Authorization

Authentication: who am I?

Authorization: what am I permitted to do?

Distributed authentication is a “solved problem”.

There are a lot of decentralized identifier (DID) projects, hence the quotes.

Authentication

Distinguish between cryptographic authentication and endowment.

Cryptographic authentication proves that an entity (machine) presenting a public key (fingerprint, identifier) also has access to the corresponding private key via challenge-response protocol.

Endowment associates identifying metadata with a public key via a signature from a trusted third party. DID projects tend to focus on endowment.

Authentication

Endowment has two phases:

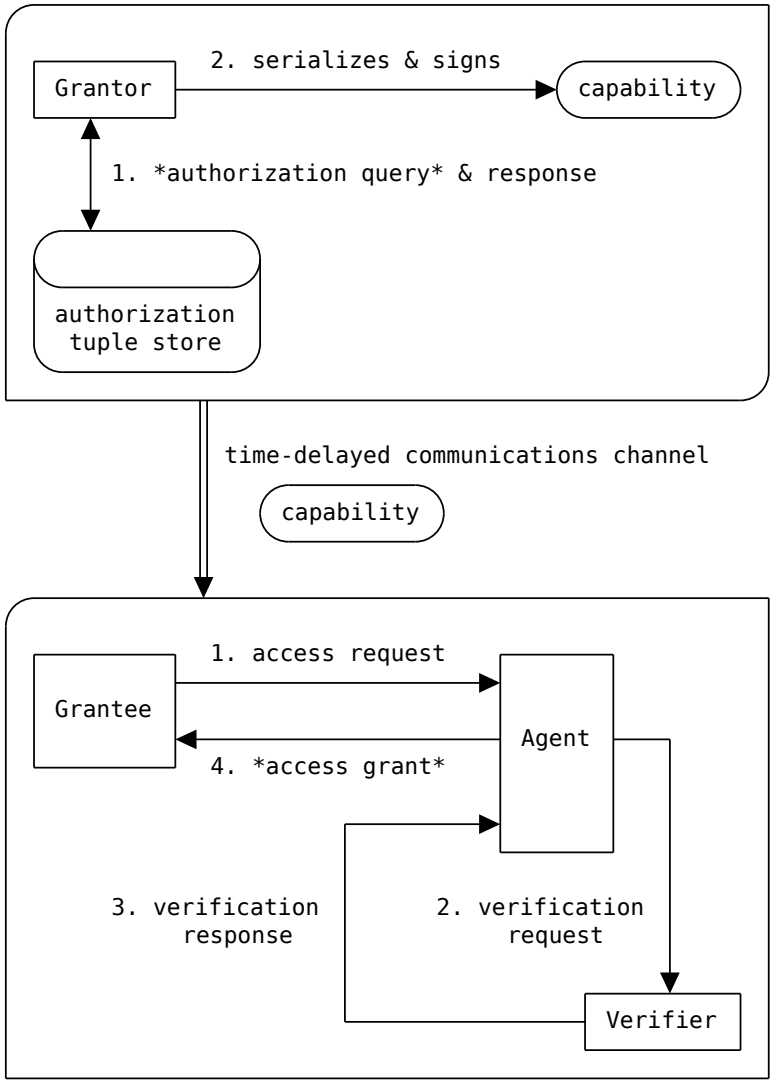
First the trusted entity signs the combination of public key and metadata (hopefully after verification). This creates a [X.509] certificate.

After (somewhat) arbitrary time, an entity that receives the above verifies the signature. The trusted entity does not need to be reachable for this (establishing this trust may need it, though).

Can we do the same for authorization?

Overview

1) Authorization occurs in advance, generates capability.
2) Capability can be stored and transmitted freely and with arbitrary delay.
3) When access is requested, the capability can be consulted to resolve whether access should be granted.



Origins

“Object capabilities” (OCAP) invented in the 1960s, and popularized with OOP.

“Identity capabilities” (ICAP) ties this to a specific bearer in the 1980s.

Pretty Good Privacy provides cryptographic model and time-delay (but is only adjacently concerned with authorization via “trust levels”).

Generalized concept: power of attorney, going back millennia.

Authorization Token Anatomy

Authorization asks: **who** is allowed to do **what** with **which** object?

who = grantee (subject)

what = privilege (predicate)

which = object

(grantee, privilege, object) tuple's ("claim") presence or absence provides binary indicator whether an action is permitted.

[Side note: could be expressed as RDF triples.]

Synchronous Process

Claim is stored in an authorization database (e.g. permission bits/ACLs in a file system, LDAP database, etc.).

When authorization should be granted, the database is consulted.

This requires **trust in** and **availability of** the database.

Asynchronous Process

Bearer presents claim with a request.

Receiver verifies the claim based *only* on the data embedded in it.

So what data is needed other than the claim tuple itself?

Distribution Requirements

Add signature from grantee, which should also be identified.

Add metadata: X.509 has e.g. “not-before” and “not-after” restrictions.

The signature is required to distribute the proof that the triple is valid. Metadata is required in practical systems, and permits management of privileges.

CAProck Goals

Make capabilities small enough to enable 0-RTT authorization (assuming 0-RTT authentication exists).

In practice, a capability then must fit in less than a single, unfragmented UDP datagram.

UDP datagrams on WiFi may fragment at above 1200 Bytes.

Capability < 1200 Bytes (in practice maybe ~500).

But: CAProck itself is flexible with regards to encoding, it just **also** defines a compact encoding, and is designed to support this compactness.

CAProck Constraints #1

Grantee, grantor identifiers are DSA/RSA SHA-3 fingerprints OR ed25519 OR ed448 public keys (max: 64 Bytes).

Object identifiers: same length, arbitrary content.

Predicate (privilege): (almost) arbitrary length and content, but for compactness be brief. Claims can be < 200 Bytes.

Signature: EdDSA defines algorithms; use SHA-2/SHA-3 for DSA/RSA. Again, this is at most 64 Bytes.

One grantor, one or more claims, one signature per capability.

CAProck Constraints #2

Metadata: validity “from” is mandatory, “to” optional (though recommended). Compact encoding uses TAI64 encoding.

Counter for resolving conflicts between information contained in multiple capabilities.

Compact encoding is self-describing, using a sequence of binary fields. Each field is prefixed by a ULEB128 field tag specifying its meaning **and** field encoding, but in practice, no more than a single Byte is used.

CAProck Library

- C API
- built using meson (https://mesonbuild.com)
- (dev) conan packages (https://conan.io)
- Forge: https://codeberg.org/interpeer/caprock/
- Specs: https://specs.interpeer.io/

Specifications

- Capabilities for Distributed Authorization
  - https://specs.interpeer.io/draft-jfinkhaeuser-caps-for-distributed-auth/
  - https://datatracker.ietf.org/doc/draft-jfinkhaeuser-caps-for-distributed-auth/
- CAProck Distributed Authorization Scheme
  - https://specs.interpeer.io/draft-jfinkhaeuser-caprock-auth-scheme/
- CAProck Compact Wire Encoding
  - https://specs.interpeer.io/draft-jfinkhaeuser-caprock-enc-compact/
- Further/future documents:
  - JSON Web Token based encoding
  - X.509 based encoding (?)

Interpeer Project

- Web: https://interpeer.io/
- Code: https://codeberg.org/interpeer/
- Mailing: https://lists.interpeer.io/
- We’re a non-profit: https://interpeer.io/donations/

THANK YOU

INTERPEER